Preprint
UCRL-JC-140114

VisIt: A Component Based Parallel Visualization Package

S. Ahern, K. Bonnell, E. Brugger, H. Childs, J. Meredith, B. Whitlock

This article was submitted to
Nuclear Explosives Code Developers Conference, Oakland, CA.,
October 23-27, 2000

October 1, 2000

U.S. Department of Energy
Lawrence
Livermore
National
Laboratory

This work was performed under the auspices of the United States Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

VisIt: A Component Based Parallel Visualization Package (U)

Sean Ahern, Kathleen Bonnell, Eric Brugger,
Hank Childs, Jeremy Meredith, Brad Whitlock
Lawrence Livermore National Laboratory

We are currently developing a component based, parallel visualization and graphical analysis tool for visualizing and analyzing data on two- and three-dimensional (2D, 3D) meshes. The tool consists of three primary components: a graphical user interface (GUI), a viewer, and a parallel compute engine. The components are designed to be operated in a distributed fashion with the GUI and viewer typically running on a high performance visualization server and the compute engine running on a large parallel platform. The viewer and compute engine are both based on the Visualization Toolkit (VTK), an open source object oriented data manipulation and visualization library. The compute engine will make use of parallel extensions to VTK, based on MPI, developed by Los Alamos National Laboratory in collaboration with the originators of VTK. The compute engine will make use of meta-data so that it only operates on the portions of the data necessary to generate the image. The meta-data can either be created as the post-processing data is generated or as a pre-processing step to using VisIt. VisIt will be integrated with the VIEWS' Tera-Scale Browser, which will provide a high performance visual data browsing capability based on multi-resolution techniques. (U)

Keywords: visualization, distributed computing, components

Introduction

VisIt's primary design goal is to quickly and efficiently visualize two- and three-dimensional tera-scale datasets from both structured and unstructured meshes. Visit incorporates several architectural features to meet these goals including a distributed architecture, a demand driven execution model, and a parallel compute engine. Secondary design goals include the ability to easily leverage the work of others and to deliver new functionality quickly. These secondary goals are being accomplished through the use of a component architecture and by basing all the visualization and data processing capabilities on the Visualization Toolkit (VTK).

The distributed architecture allows the different tasks required to generate a visualization to be performed on the hardware platform most appropriate to the task. The I/O can be performed on the machine where the data is located eliminating the need to move the data. The data processing and generation of the geometric primitives can be performed on a large parallel compute platform where the greatest compute capabilities are available. Finally, the rendering can be performed on a visualization server, which contains hardware for transforming geometric primitives into an image.

A demand driven execution model allows VisIt to read in and process only the data that will actually contribute to a given image. VisIt will make use of meta-data to eliminate portions of the dataset that will not be visible in the final image or that will be eliminated as part of the data processing being performed to generate the image. The datasets must be broken up into domains with the meta-data defined down to the level of a domain. Portions of the dataset will be eliminated down to the granularity of a domain.

Visualization capability developed in VTK can be easily integrated within VisIt by virtue of the fact that its architecture is based on it. With the exception of user interface development, adding a VTK module will be a straightforward process that can be automated. Functionality that does not readily lend itself to incorporation within the VTK based architecture may still be integrated with VisIt by first making it into a component.

VTK

The Visualization Toolkit is an object oriented visualization system. It contains extensive image processing and visualization capabilities. Its data model supports images, geometric primitives, structured and unstructured grids, and scalar, vector and tensor fields defined on grids. The VTK library consists of hundreds of modules with various visualization capabilities. The modules are combined into networks, which are then executed to form images. VTK is written in C++ and runs under Unix, Windows95 and WindowsNT. It also supports interfaces for Tcl/Tk and Python.

Architecture

VisIt consists of three primary components including a Graphical User Interface (GUI), a Viewer, and a parallel Compute Engine. Additionally, there is a Meta-data Server, which provides file system information as well as meta-data about the datasets being visualized. The components interact with one another through the use of proxy classes. The proxy classes are responsible for starting the component or connecting to an existing component. The proxy classes are then used to interact with the components.

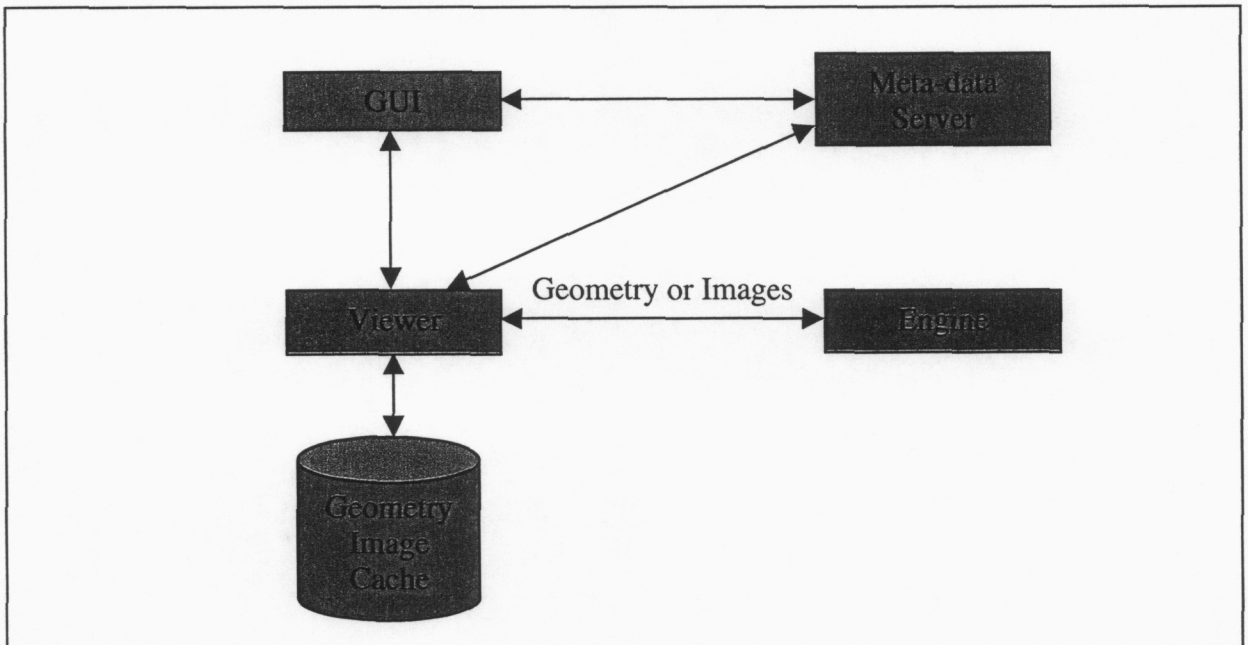

Figure 1. A block diagram of the VisIt components.

The state information for VisIt is shared between the GUI and the Viewer, with the majority of the state information stored in the Viewer. The state information is kept consistent between the GUI and Viewer through the use of the Subject Observer design pattern. The Meta-data Server

and the Compute Engine are both stateless except for any internal caching they may be doing. This leads to a highly fault tolerant architecture. If the machine that the Meta-data Server and Compute Engine was running on were to go down, the Viewer could automatically restart them later and continue from where the user left off.

The Viewer and Compute Engine create dynamic VTK networks as necessary to generate the images displayed by the Viewer. The Compute Engine will either send geometry or images to the Viewer. Currently only geometry is sent between the Compute Engine and Viewer. The decision as to which will be user controllable and will be based on several constraints including

- The bandwidth between the Compute Engine and the Viewer
- The size of the geometric data
- The size of the image data
- The rendering speed of the Viewer
- The rendering speed of the Compute Engine

The geometric data grows with image complexity while the image data is fixed for a given image size. Since the bandwidth between the Compute Engine and Viewer is fixed and that the rendering speed of the Compute Engine scales with the number of processors, sending images will be preferred method for the most complex images. Switching between geometry and images could at some later date be made automatic.

The Viewer can connect to multiple Compute Engines. Currently it blocks waiting for data to arrive from the Compute Engine but in the future will be threaded so that it can perform other operations while it is waiting for data. The Viewer caches the geometric primitives for handling viewing changes and for displaying time-based animations. In the future, the Viewer will also cache images for displaying time-based animations.

The Compute Engine can operate in serial of parallel. The parallel version uses the Message Passing Interface (MPI) for doing its parallel communication. Datasets must be decomposed into domains for the parallel version to achieve any type of performance gain over the serial version. The parallel version currently employs a static load-balancing scheme, where domains are assigned to processors before any work is done. In the future it will be employing a dynamic load-balancing scheme where the domains are placed in a queue and are assigned to processors when they become idle. The Compute Engine caches the last ten most recently used networks. It provides progress information to the Viewer at the granularity of a domain. It is also interruptible, again at the granularity of a domain.

The component architecture allows other applications to easily be integrated within VisIt. The VIEWS' Tera-Scale Browser and the VIEWS' Scientific Data Management (SDM) feature extraction tools are both components we are planning on integrating with VisIt. Figure 2 illustrates how the components will work together. The Tera-Scale Browser and VisIt will share a common GUI, with appropriate extensions added to the existing interface to handle the additional functionality that the Tera-Scale Browser supplies.

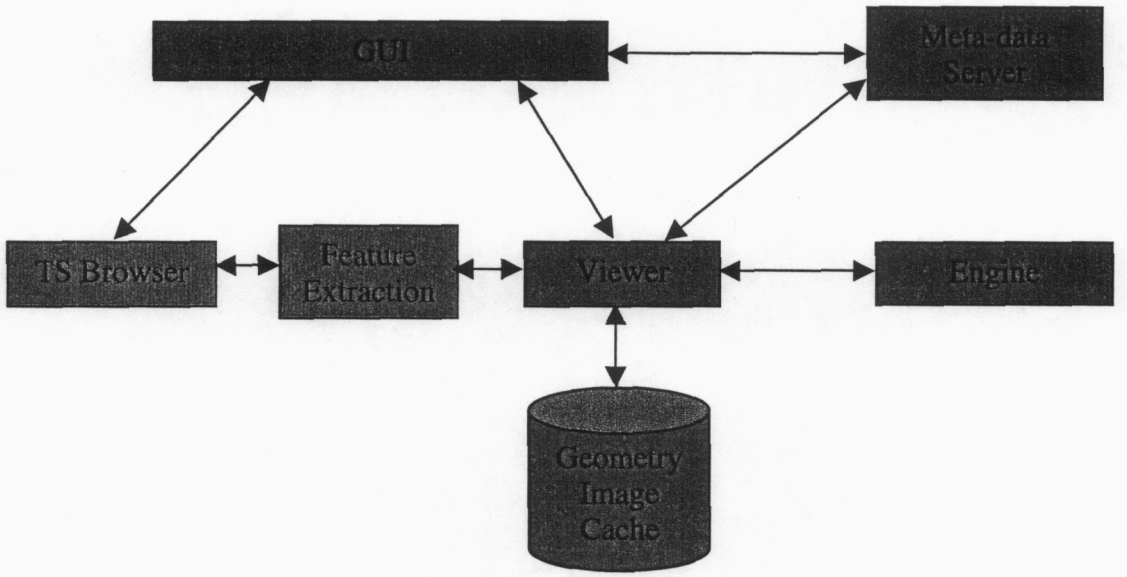

Figure 2. A block diagram showing the Tera-Scale Browser and SDM feature extraction tools integrated within VisIt.

Current Status

We have focused on creating the infrastructure for VisIt and have implemented the minimal amount of visualization capability necessary to develop and test the infrastructure. It currently supports both two- and three-dimensional structured and unstructured data. As of October 1, 2000, VisIt had the ability to

- Create pseudocolor plots
- Slice and onion peel datasets
- Seamlessly access data on remote machines
- Launch remote engines as necessary to create plots
- Create geometry on a remote machine and send it to the viewer for display
- Display images in multiple windows
- Create multiple plots
- Apply multiple operators to a dataset
- Set plot and operator attributes
- Hide and show plots
- Animate time dependent data
- Post plot and operator attribute windows to a tabbed notepad

VisIt has been ported to SGI and Linux systems.

Screen captures from VisIt

Below are a couple of screen captures from VisIt displaying some of the functionality currently available in VisIt.

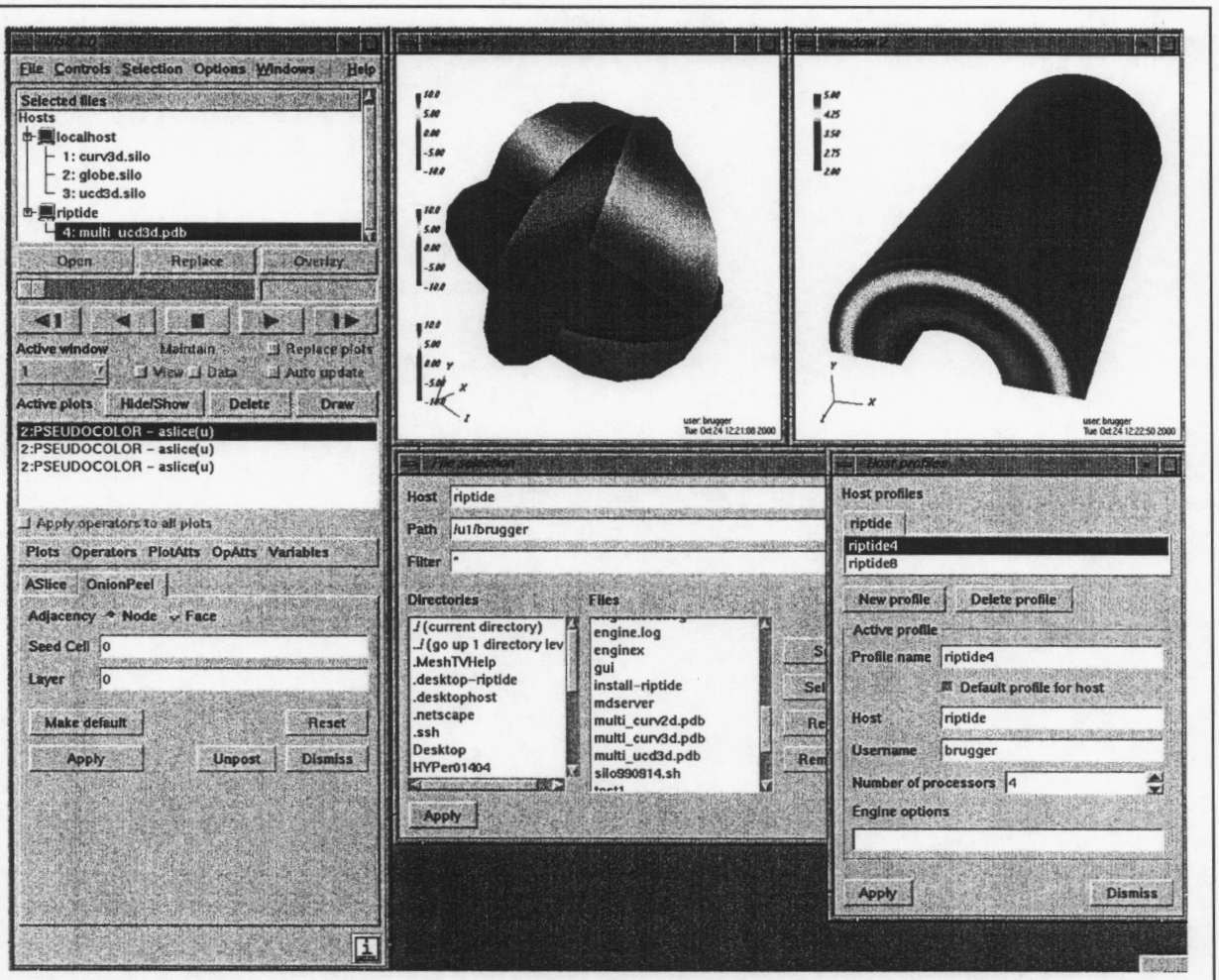

Figure 3. A screen capture of VisIt displaying data from two remote machines. The upper left hand window contains an image from a Compute Engine running serially on the same machine as the GUI and Viewer. The upper right hand window contains an image from a Compute Engine running in parallel on a remote machine. Also shown is the Host profiles window in the lower right hand corner. It is used to set the properties for the Compute Engines that are initiated by VisIt, such as the number of processors to use or the user name on the remote machine.

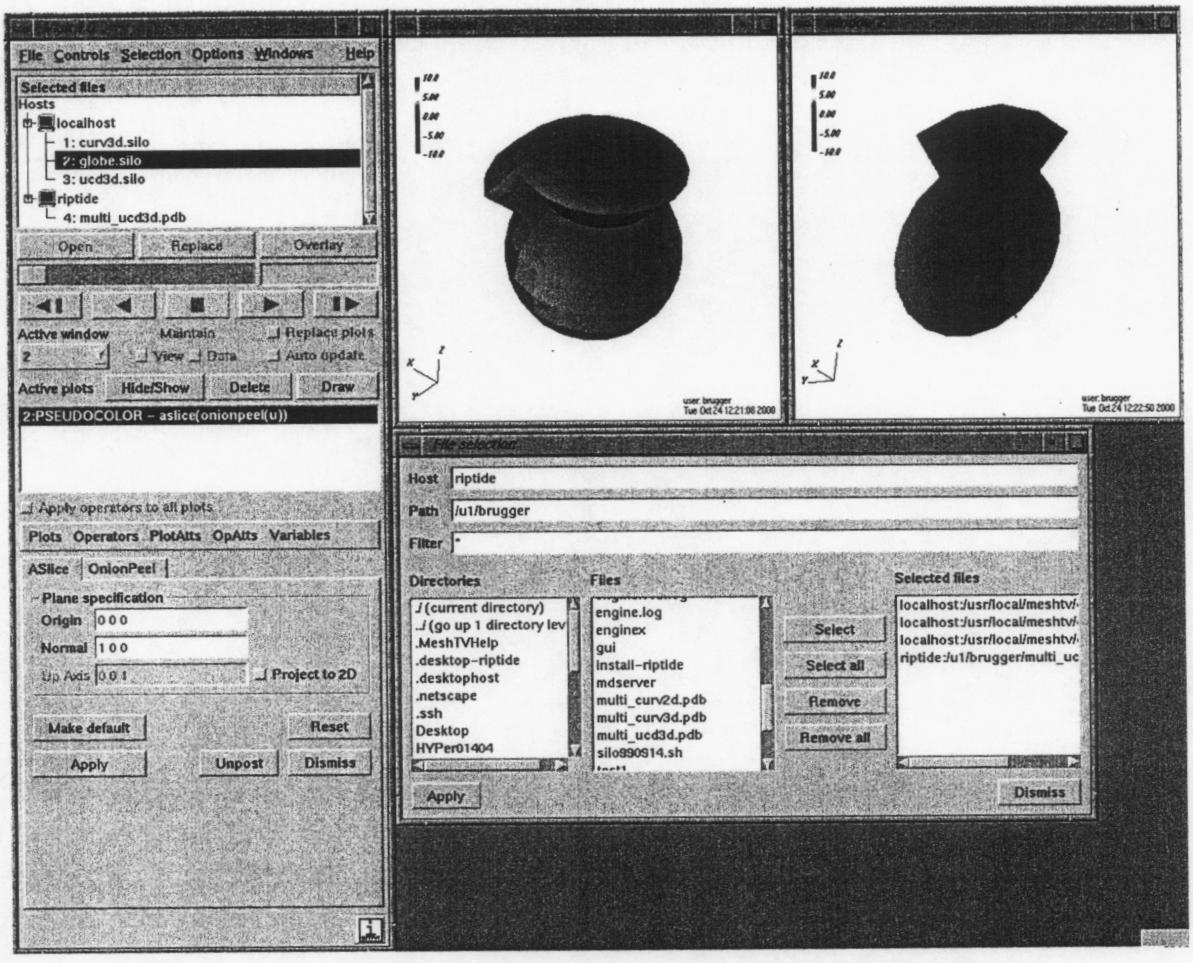

Figure 4. A screen capture of VisIt displaying data with multiple operators applied to the datasets. The upper left hand window shows a 3D unstructured mesh with the onion peel operator applied. The upper right hand window shows the onion peeled mesh sliced by a cutting plane.

Future Plans

We will continue focusing on completing the infrastructure. We will then implement a base level of visualization capabilities for a wide spread release in July 2001.

References

Schroeder, W., Martin, K., and Lorensen, B., *The Visualization Toolkit,* (Prentice Hall PTR, New Jersey, 1998).

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns,* (Addison-Wesley, Massachusetts, 1995).

Acknowledgements

This work was performed under the auspices of the U. S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.